FMS: the GFDL Flexible Modeling System

V. Balaji
SGI/GFDL

OASIS Workshop
CERFACS, Toulouse
19 October 2000

GFDL

GFDL is a climate modeling centre. The primary focus is the use of coupled climate models for simulations of climate variability and climate change on short and long time scales.

Current computing capability: Cray T90 24p, T3E 128p.

Future computing capability: $8 \times 128 + 2 \times 64$p Origin 3000.

GFDL models

- MOM: Modular Ocean Model.
- FMS: Flexible Modeling System.
- Hurricane model.
- HIM: isopycnal model.
- 2 non-hydrostatic atmospheric models.
- Older models: SKYHI, Supersource.

Modernization

- Parallelism without compromising vector performance.

- Modular design for interchangeable dynamical cores and physical parameterizations. Several dynamical cores are currently available.

- Distributed development model: many contributing authors. Use high-level abstract language features for encapsulation, polymorphism.

FMS: Flexible Modeling System

Jeff Anderson, Balaji, Paul Kushner, Ron Pacanowski, Bruce Wyman, ...

Dynamical cores:

- Atmosphere:
  - Hydrostatic spectral
  - Hydrostatic Arakawa B grid
  - Hydrostatic Arakawa C grid (*)
  - Non-hydrostatic Arakawa C grid (*)

- Ocean:
  - B grid
  - C grid (*)
  - Generalized vertical coordinate (*)

FMS: Physical processes

- Atmosphere:
  - Deep convection.
  - Shallow convection.
  - Moist processes.
  - Cloud mass flux.
  - Ozone, CFCs, greenhouse gases.
  - Radiation.
  - Turbulence.
  - Planetary boundary layer.
  - Land surface, ocean surface.

Elements of FMS

FMS consists of:

component models code describing the evolution of a climate model sub-system: atmosphere, ocean, land, ice, ... also often called **dynamical cores**.

drivers *coupled* and *solo*.

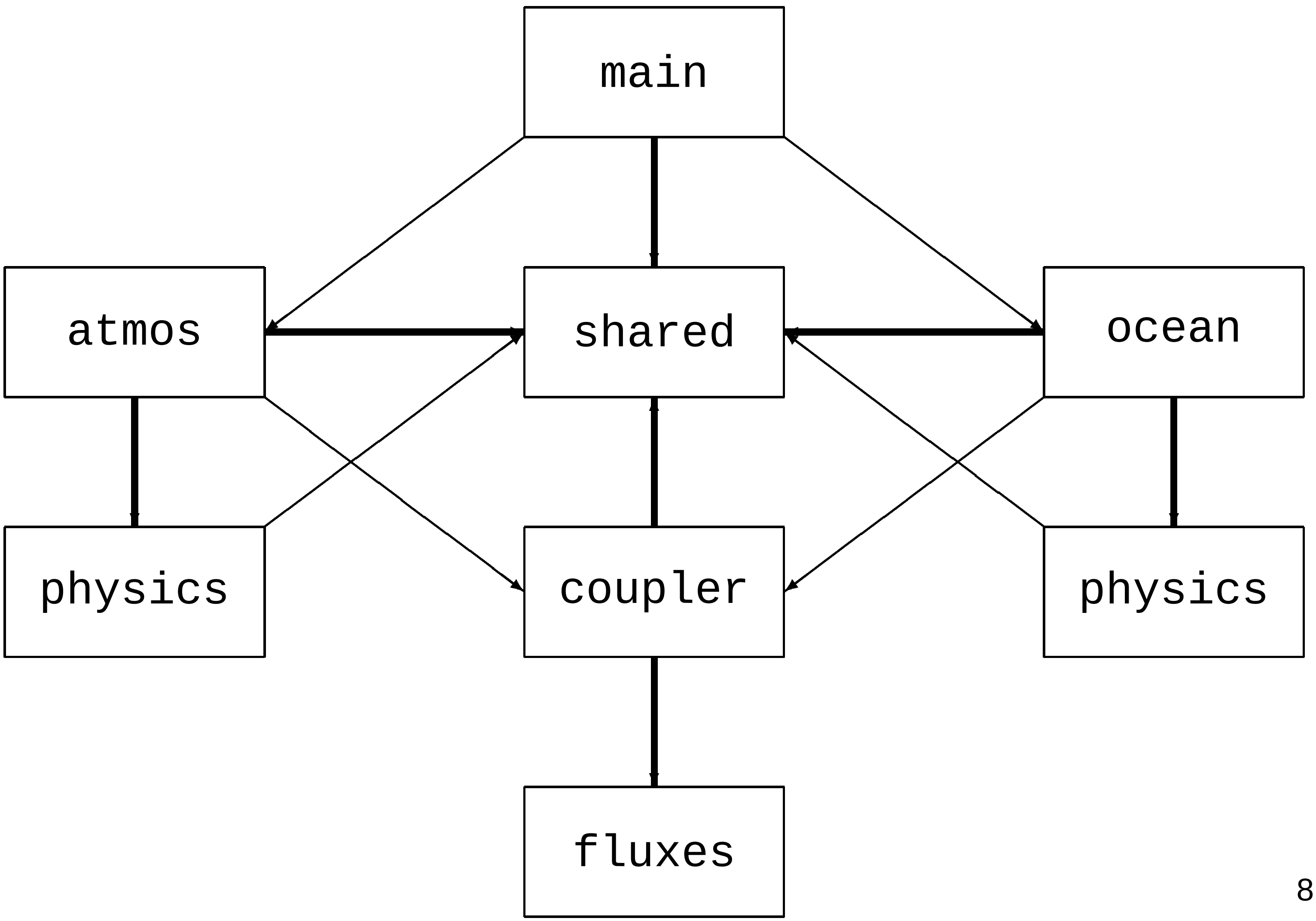
parameterizations physics routines.

coupler routine for exchanging data at model boundaries.

Features of FMS

- FMS runs as a single executable.
- Dynamical cores for a particular climate subsystem component present a uniform boundary interface.
- Component models may run serially or simultaneously.
- Standard interface for column physics.
- Shared code for parallelism, I/O, diagnostics, calls to standard scientific libraries.

FMS calling structure



Shared code

- MPP modules: communication kernels, domain decomposition and update, parallel I/O.

- Diagnostics handler: diagnostic registry, call by alarm.

```
id = register_diag_field( ... )
```

- Scientific libraries.

```
real :: grid(:,:,:)
complex :: fourier(:,:,:)
fourier = fft(grid)
```

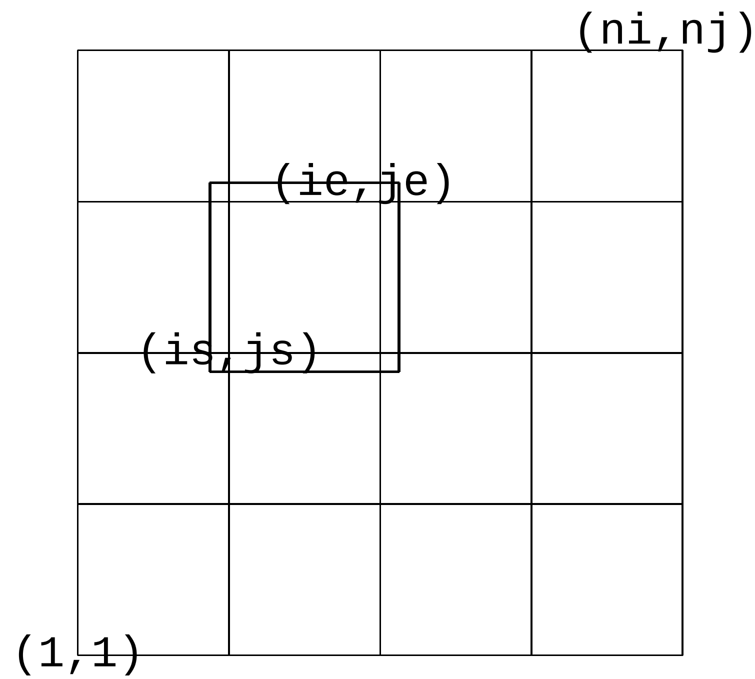
Parallel programming interface

GFDL has a homegrown parallelism API written as a set of 3 F90 modules:

- `mpp_mod` is a low-level interface to message-passing APIs (currently SHMEM and MPI; MPI-2 and Co-Array Fortran to come);
- `mpp_domains_mod` is a set of higher-level routines for domain decomposition and domain updates;
- `mpp_io_mod` is a set of routines for parallel I/O.

`http://www.gfdl.gov/~vb`

```
!domaintypes of higher rank can be constructed from type domain1D
  type, public :: domain2D
     sequence
     type(domain1D) :: x
     type(domain1D) :: y
     integer :: pe
     type(domain2D), pointer :: west, east, south, north
  end type domain2D
```



mpp_domains_mod calls:

- `mpp_define_domains()`

- `mpp_update_domains()`

```
type(domain2D) :: domain(0:npes-1)
call mpp_define_domains( (/1,ni,1,nj/), domain, xhalo=2, yhalo=2 )
...
!allocate f(i,j) on data domain
!compute f(i,j) on compute domain
...
call mpp_update_domains( f, domain(pe) )
```

Parallel I/O

```
type(domain2D) :: domain(0:npes-1)
type(axistype) :: x, y, z, t
type(fieldtype) :: field
integer :: unit
character*(*) :: file
real, allocatable :: f(:,:,:)
call mpp_define_domains( (/1,ni,1,nj/), domain )
call mpp_open( unit, file, action=MPP_WRONLY, format=MPP_IEEE32, &
  access=MPP_SEQUENTIAL, threading=MPP_MULTI, fileset=MPP_MULTI )
call mpp_write_meta( unit, x, 'X', 'km', ... )
...
call mpp_write_meta( unit, field, (/x,y,z,t/), 'Temperature', 'kelvin', ... )
...
call mpp_write( unit, field, domain(pe), f, tstamp )
```

`mpp_io_mod` output modes

`mpp_io_mod` supports three types of parallel I/O:

- Single-threaded I/O: a single PE acquires all the data and writes it out.
- Multi-threaded, single-fileset I/O: many PEs write to a single file.
- Multi-threaded, multi-fileset I/O: many PEs write to independent files (requires post-processing).

Coupler

Used for the exchange of fluxes between models. Key features include:

Conservation: required for long runs.

Resolution: the coupler places no constraints on component model timesteps and spatial resolution. Supports both explicit and implicit timesteps, and the exchange computation is not rate-limiting.

Exchange grid: union of component model grids, where detailed flux computations are performed (Monin-Obukhov, tridiagonal sover for implicit diffusion, ...)

Fully parallel: Calls are entirely processor-local: exchange software will perform all interprocessor communication.

Modular design: uniform interface to main calling program.

No brokering: each experiment must explicitly set up field pairs.

Single executable.

Implicit timestepping

```
type (atmos_boundary_data_type) :: Atm
type (ocean_boundary_data_type) :: Ocean
type  (land_boundary_data_type) :: Land
type   (ice_boundary_data_type) :: Ice
do no = 1, num_ocean_calls
   call flux_ocean_to_ice (Ocean, Ice, ... )
   call ice_bottom_to_ice_top (Ice, ... )
   do na = 1, num_atmos_calls
      Time = Time + Time_step_atmos
      call update_atmos_model_down  (Atm, ... )
      call flux_down_from_atmos (Time, Atm, Land, Ice, ... )
      call update_land_model_fast (Land, ... )
      call update_ice_model_fast (Ice, ... )
      call flux_up_to_atmos (Time, Land, Ice, ... )
      call update_atmos_model_up   (Atm,  ... )
   enddo
   call update_ice_model_slow (Ice, ... )
   call flux_ice_to_ocean ( Ice, ... )
   call update_ocean_model (Ocean, ... )
```

Coupler example

```
subroutine flux_down_from_atmos (Time, Atm, Land, Ice, ... )
  type (atmos_boundary_data_type), intent(in)  :: Atm
  type  (land_boundary_data_type), intent(in)  :: Land
  type   (ice_boundary_data_type), intent(in)  :: Ice
  call put_exchange_grid (Atm%flux_sw,  ex_flux_sw,  bd_map_atm)
  call put_exchange_grid (Atm%flux_lw,  ex_flux_lwd, bd_map_atm)
  call gcm_vert_diff_surf_down ( ... )
  call get_exchange_grid (ex_flux_sw, flux_sw_land, bd_map_land)
  call get_exchange_grid (ex_flux_lw, flux_lw_land, bd_map_land)
  call get_exchange_grid (ex_flux_sw, flux_sw_ice, bd_map_ice_top)

end subroutine flux_down_from_atmos
```

Future directions

- Testing and evaluation on a variety of coupled scenarios.

- Production on various systems, internal and external; external support on a collaborative basis.

- Use of abstract distributed field datatypes for generic numeric kernels on multiple stencils and grids.

Parallel numerical kernels

$$\frac{\eta^{n+1} - \eta^n}{\Delta t} = -H(\nabla \cdot \mathbf{u})^n \qquad (1)$$

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = -g(\nabla \eta)^{n+1} + f\mathbf{k} \times \left(\frac{\mathbf{u}^{n+1} + \mathbf{u}^n}{2}\right) + \mathbf{F} \qquad (2)$$

```
program shallow_water
  type(scalar2D) :: eta(0:1)
  type(hvector2D) :: utmp, u, forcing
  integer tau=0, taup1=1
...
  f2 = 1./(1.+dt*dt*f*f)
  do l = 1,nt
     eta(taup1) = eta(tau) - (dt*h)*div(u)
     utmp = u - (dt*g)*grad(eta(taup1)) + (dt*f)*kcross(u) + dt*forcing
     u = f2*( utmp + (dt*f)*kcross(utmp) )
     tau   = 1 - tau
     taup1 = 1 - taup1
  end do
end program shallow_water
```